



REGULAR PAPER

Daniel Limberger · Willy Scheibel · Jan van Dieken · Jürgen Döllner

Procedural texture patterns for encoding changes in color in 2.5D treemap visualizations

Received: 3 February 2022 / Revised: 25 July 2022 / Accepted: 19 August 2022
© The Author(s) 2022

Abstract Treemaps depict tree-structured data while maintaining flexibility in mapping data to different visual variables. This work explores how changes in data mapped to color can be represented with rectangular 2.5D treemaps using procedural texture patterns. The patterns are designed to function for both static images and interactive visualizations with animated transitions. During rendering, the procedural texture patterns are superimposed onto the existing color mapping. We present a pattern catalog with seven exemplary patterns having different characteristics in representing the mapped data. This pattern catalog is implemented in a WebGL-based treemap rendering prototype and is evaluated using performance measurements and case studies on two software projects. As a result, this work extends the toolset of visual encodings for 2.5D treemaps by procedural texture patterns to represent changes in color. It serves as a starting point for user-centered evaluation.

Keywords Treemaps · Visual variables · Animated transitions · Change visualization

1 Introduction

Treemaps allow for the visualization of tree-structured data while supporting mappings to multiple visual variables, including size and color (Johnson and Shneiderman 1991; Carpendale 2003). They are applied for visualization in domains such as business intelligence and software development and facilitate data observation over time (Roberts and Laramee 2018; Caserta and Zendra 2011). Treemaps can be used in static and interactive contexts, i.e., using static images or interactive clients to support interactive visual analytics tasks. In contrast to 2D treemaps, their embedding in the third dimension, so-called *2.5D treemaps* or *3D-embedded treemaps*, enable additional data mapping and facilitate a city-like metaphor often suggested for the depiction of software projects (Wettel and Lanza 2008; Schulz et al. 2011; Limberger et al. 2019b). Although different layouting shapes exist for treemaps (Scheibel et al. 2020), we focus on the widely used rectangular layouts, and, thus, rectangular 2.5D treemaps.

D. Limberger (✉) · W. Scheibel · J. van Dieken · J. Döllner
Hasso Plattner Institute, Digital Engineering Faculty, University of Potsdam, Potsdam, Germany
E-mail: daniel.limberger@hpi.uni-potsdam.de

W. Scheibel
E-mail: willy.scheibel@hpi.uni-potsdam.de

J. van Dieken
E-mail: jan.dieken@hpi.uni-potsdam.de

J. Döllner
E-mail: juergen.doellner@hpi.uni-potsdam.de

Published online: 03 October 2022

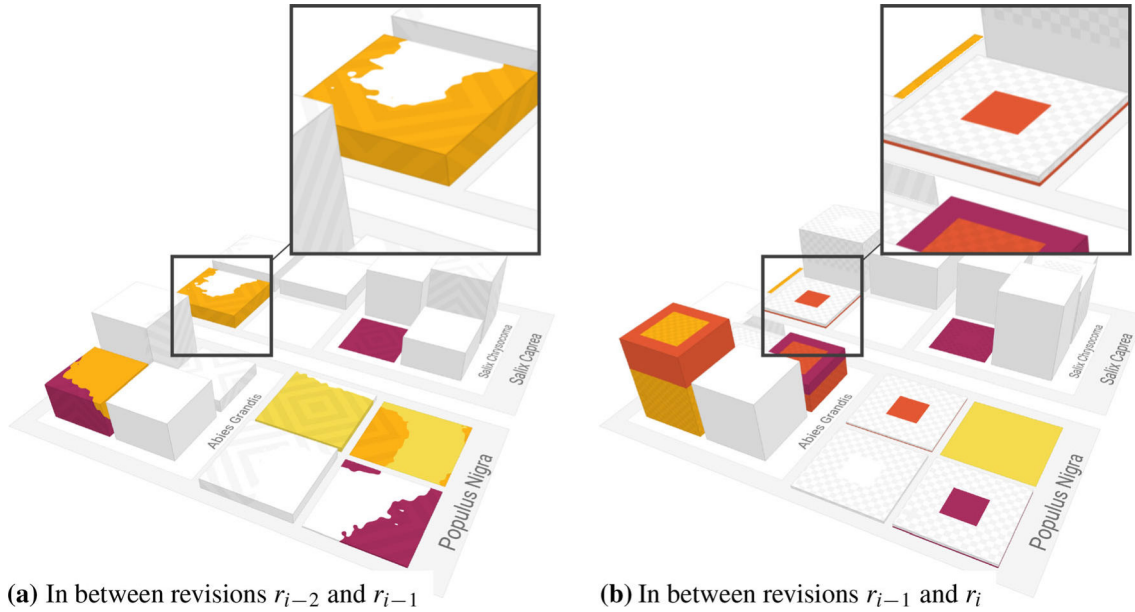


Fig. 1 Synthetic hierarchical test dataset with n revisions is depicted using a 2.5D treemap that supports transitioning between subsequent data revisions mapped to color and height. The two examples **a** and **b** show a combination of two procedural patterns used to animate transitions between revisions

In order to depict changes in time-varying data, we use an abstraction of a former and a latter state, whose values should be encoded at their respective nodes. This can be achieved by depicting the former and latter state using multiple visual variables simultaneously or extending the capabilities of existing ones, e.g., using in-situ representations (Tu and Shen 2007) or procedural textures. This idea was recently applied to 2.5D treemaps to depict changes in multiple attributes simultaneously (Limberger et al. 2019a).

When taking advantage of the additional visual variables offered in three dimensions, an important task is to use an effective encoding that allows identifying the former state and the latter state, as well as the magnitude and the direction of a change in values. We identify these as the *direction of animation*, the *magnitude of change*, and the *direction of change*, respectively. In the case of the 2D contrast treemap of Tu and Shen, this is done by explicit ordering and fixed orientation of the treemap, allowing for an easy-to-replicate reading direction. For 2.5D treemaps, procedural textures can be used to encode the direction of change for the spatial dimensions, such as a change in height. However, encoding the direction of change for the color mappings is incompatible with this approach.

In this work, we extend on an earlier work using procedural patterns to depict data changes over time in 2.5D treemaps (Limberger et al. 2021). Based on this, we present an approach to design and integrate procedural texture patterns for rectangular 2.5D treemaps with area mapping, height mapping, and color mapping while focusing on communicating changes in color. The patterns are suitable for both static and interactive application contexts, i.e., used as a static image or during animated transitions, respectively (Fig. 1). We showcase the approaches using a prototypical implementation in WebGL that can visualize treemaps with up to hundreds of thousands of nodes. Extending on the previous work, this includes details on our implementation, further applications, and a discussion. In summary, our main contributions are:

- the concept of procedural texture patterns that are suitable for mapping change in color in rectangular 2.5D treemaps,
- a catalog of seven specific patterns as examples to showcase the design space and the trade-off between characteristics,
- an extension to include a secondary patterns for accurate display of change direction even for static display of an image,
- a detailed description of a prototypical GPU-based implementation with focus on memory layout of per-vertex data and the integration of the procedural textures in fragment shader calls, and

- an evaluation of the prototype with focus on performance impact compared to a rendering system without texture pattern capabilities, and two case studies on real-world software projects.

The remainder of this paper is structured as follows: Section 2 presents and discusses related work. Section 3 introduces procedural textures to extend the color mapping of 2.5D treemaps and a catalog of example patterns. Section 4 contains evaluation by means of a descriptions of our prototype, the case studies, and a discussion. Last, Sect. 5 concludes this paper.

2 Related work

Depicting change in data is an often-performed mapping in InfoVis and not novel to treemaps. Although there are more aspects to adopting treemaps for use over time, such as layout stability (Vernier et al. 2020), we focus on depicting change within this work. Typically, changes can be depicted using comparative layouts, such as juxtaposition, superposition, explicit encoding, and animated transition (L’Yi et al. 2021). With software maps, an example application of 2.5D treemaps, we can represent aspects of a software system for a specific revision or time. We want to enable users to interactively explore correlations in time-varying software system engineering data, e.g., metrics per revision, development budgets, status and progress of defects and issues, team composition, development activity, and others. This data should be depicted not only for a specific point in time but over time as well.

For example, the time dimension can be incorporated by distributing multiple maps spatially using small multiples (Scheibel et al. 2016; Chen et al. 2017). Provided the evolution over more than just two states is already analyzed and available by means of trend data, natural metaphors such as physical-based material degradation or shininess can be an effective tool for communication (Würfel et al. 2015). Another goal is to augment height and color mappings already in use and rely on the user’s existing understanding of treemaps while emphasizing changes in the data. Thereby, the representation of change should visually correspond to the change in data (*visual-data-correspondence*) (Kindlmann and Scheidegger 2014). This holds for both the actual difference between two values and the direction of change (or sign of the difference). For this, basic variants of *data vases* (Thakur and Rhyne 2009) or *in situ templates* (Limberger et al. 2019a) can be used. The latter allows for change encoding of up to three visual variables, i.e., area, color, and height, simultaneously. However, reading direction issues must be considered, and ample training is required, too. An example for the use of *in situ templates* to display changes in height is given in Fig. 2.

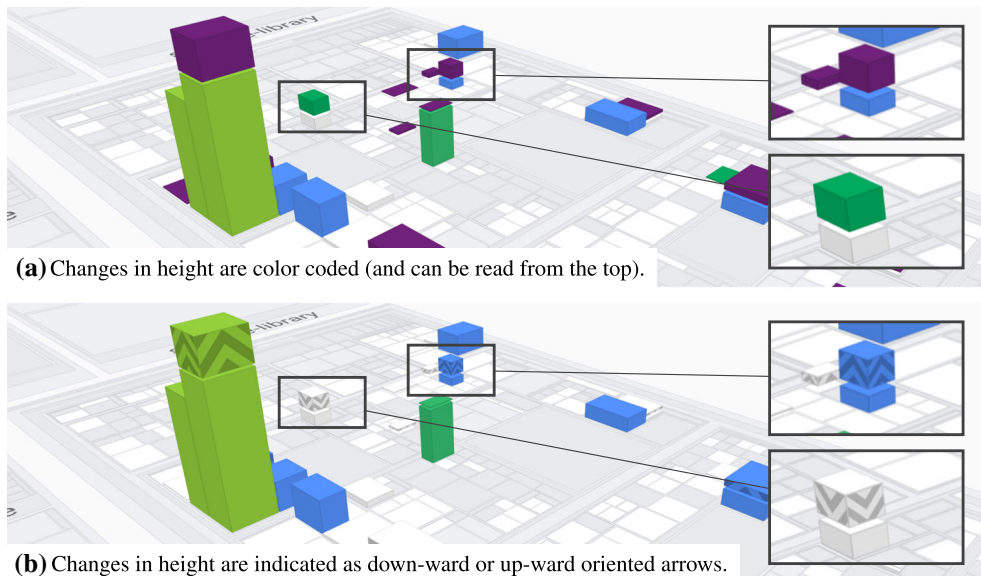


Fig. 2 Basic approaches for communicating changes of subsequent data values mapped to height using colors (top) and arrows (bottom) mapped to the actual difference in height. The former uses a separate color mapping, thereby partially occluding the actual color mapping, and the latter uses procedural patterns for explicit encoding of the change in height and the change direction using an arrow pattern

These approaches, however, handle the depiction of change in color by increasing the geometric complexity. In contrast, we extend on approaches that encode the change into the existing geometric representation. For 2D treemaps, color patterns such as *two corners*, resulting in a diagonal color gradient, or contrast modifications like *ratio shading* can be used for 2D treemaps (Tu and Shen 2007). Regarding 2.5D treemaps, animation using an interpolation of height and color attributes could be used, too (Bladh et al. 2005; Langelier et al. 2008). With our focus on changes in data mapped to color, our approach is also related to encoding multiple colors within the same surface or space, known as color weaving (Urness et al. 2003), which was also applied to visualize two measures in color on treemaps (Benomar et al. 2013). For this weaving, the effectiveness of reconstructing the ratio of colors as a user was examined for 2D visualizations (Hagh-Shenas et al. 2007).

3 Procedural texture patterns

For a rectangular 2.5D treemap, a cuboid’s top face and lateral faces are typically used for single-value color mapping, which we use to superimpose a second color representing a second value. For that, we use the concept of transition-aware texture patterns that are procedurally generated using *procedural textures* in computer graphics (Quilez 2013). These texture patterns are applied to the top and lateral faces and allow to show both colors on a cuboid’s surface. Such a pattern maps a distribution of two colors onto a surface without introducing additional colors.

Let $C = [0, 1]^3 \subseteq \mathbb{R}^3$ denote the unit cube in the Euclidean space \mathbb{R}^3 that is used to provide a local parameterization of a treemap node. Its surface S is given by the boundary of C , i.e., $S = \partial C$. A pattern on the surface S is a function given by

$$P : S \times [0, 1] \rightarrow \{c_f, c_l\}, (p, \lambda) \mapsto P(p, \lambda) = P_\lambda(p),$$

where $\lambda \in [0, 1]$ is called progress value and c_f and c_l are two colors representing the value of the former state and the latter state. Furthermore, a pattern P satisfies the following three properties:

1. $\text{area}(\{P_0 = c_l\}) = 0$,
2. $\text{area}(\{P_1 = c_l\}) = \text{area}(S)$, and
3. $\forall \lambda_1, \lambda_2 \in [0, 1] : \lambda_1 < \lambda_2 \Rightarrow \text{area}(\{P_{\lambda_1} = c_l\}) \leq \text{area}(\{P_{\lambda_2} = c_l\})$,

where area denotes the usual area of surface in \mathbb{R}^3 and the set $\{P_\lambda = c_l\}$ denotes the subset points on S , where the pattern takes the value c_l .

As specific examples of such transition-aware texture patterns, we propose seven patterns, namely (1) Pillar, (2) Pyramid, (3) Arrows, (4) Arrows (full), (5) Noise, (6) Dithering, and (7) Squares (Fig. 3). As the proposed patterns share approaches but also differ in their visual display and characteristics, we propose a set of characteristics to capture their abilities for information encoding. Further, we propose an extension to superimpose two patterns to enhance the direction of change for static images. In an interactive context, this progress value λ should gradually increase from 0 to 1, resulting in an animated transition. In static contexts, this value can be chosen constant, e.g., the progress value $\lambda = 1/2$ that represents half the transition.

3.1 Pattern characteristics

A mapping from a progress value to the color distribution of two colors on a surface brings a number of optimization possibilities. We focus on five desirable characteristics that try to capture what a pattern can visually *encode* and how this pattern transforms during animated transitions, i.e., the more a texture pattern supports, the more it should be suitable for mapping change onto color. They are designed to capture whether a pattern can discern the former and the latter color (read direction), and the current transition progress value. Additionally, we add four characteristics that attribute *compatibility*. As such, they support applications of the texture patterns to treemaps in other contexts and differing parameterizations, e.g., the combination with other visual variable mappings and their potential animated transition, or choice of the base shapes for layouting. The encoding characteristics are as follows:

$\nabla_{\text{Monotonic}}$ A pattern is *monotonic* if a point of the surface switches from displaying the former color to the latter color only once. We only propose patterns in the catalog that have this characteristics. Patterns that are not *monotonic* introduce flickering into an animated transition.

$\nabla_{\text{DirStates}}$

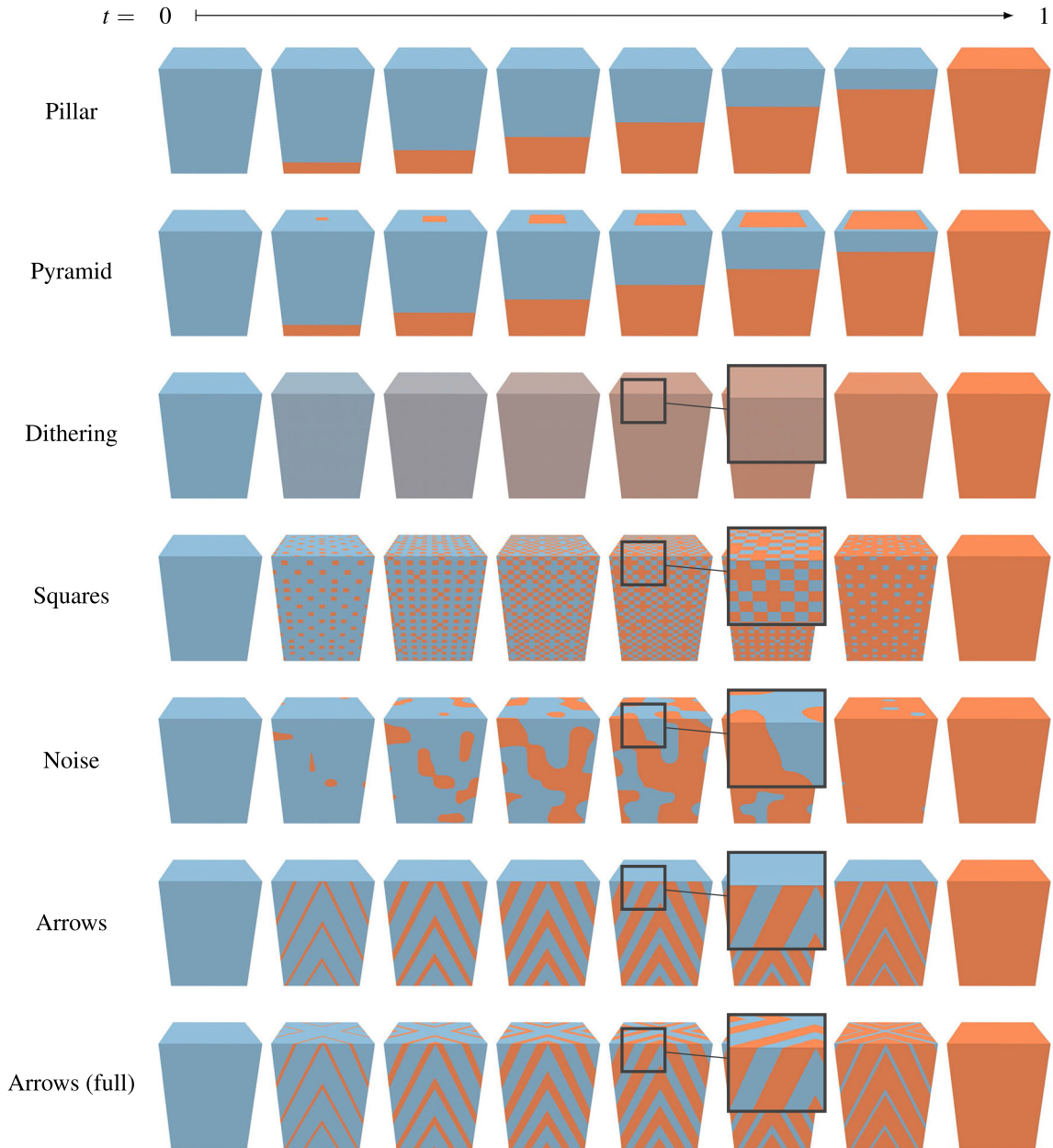


Fig. 3 Procedural patterns for changes in color attribute, from top to bottom: Pillar, Pyramid, Dithering, Squares, Noise, Arrows, and Arrows (full)

This characteristic measures whether the two encoded states can be read unambiguously, i.e., discerning the former from the latter state, in a static image or while the animation is paused during a transition. Further, this takes different viewpoints of the user as well as potential occlusion of parts of a cuboid into account.

$\nabla_{\text{DirChange}}$ The direction of change in value, increase or decrease, can be read unambiguously in a static image or while the animation is paused during a transition. This also takes different viewpoints as well as potential occlusion of parts of a cuboid into account.

∇_{RatioLat} The ratio of different colors on the lateral faces matches a transition's progress. These lateral faces' area scales with the value mapped to height and partially with the value mapped to weight.

∇_{RatioTop} The ratio of different colors on the top face matches a transition's progress (top view or 2D treemap). The top face area scales with the value mapped to weight.

The compatability characteristics are as follows:

$\nabla_{\text{IndHeight}}$	The mapping of change difference and change direction does not compromise the height mapping as well as a display of change mapped to height.
$\nabla_{\text{IndWeight}}$	The mapping of change difference and change direction does not compromise the weight mapping as well as a display of change mapped to weight.
∇_{TreeSize}	The pattern can be easily adjusted for large or very large treemaps. This characteristic holds, if for a range of treemap sizes, the pattern can be read in a meaningful way, i.e., all other characteristics are retained.
$\nabla_{\text{Polygonal}}$	The pattern is suitable to be applied to treemaps with general polygonal shapes. This includes that the pattern's visual metaphor upholds and an implementation is identical or similar to the rectangular one.

3.2 Texture patterns catalog

More technically, a texture pattern describes a visual mapping of transition progress for a fragment on a cuboid's surface using a binary per-fragment choice of color for a treemap node's former and latter state. Each pattern is designed to encode the transition progress by the area ratio of the two colors of the two depicted states. The seven patterns are a starting point and by no means a complete list of alternatives. Thereby, each pattern is not precisely defined by a specific implementation or a fragment-precise definition of the binary decision to be made but represents the overall appearance with alternatives and additional parameters.

3.3 Pillar

The pillar pattern partitions the surface vertically and assigns the former and latter colors to a top share and a bottom share of the pillar, respectively. This pattern's transition starts with a full pillar using the former color. Then, the next color *grows* from the top and overlays the former color step by step. Thus, after a brief training period, a user should be able to determine the former color and the latter color, even for a static display. However, this pattern does not show partial progress on the top faces. Instead, the color of the top face changes instantaneously with the completion of the transition.

3.4 Pyramid

The pyramid pattern extends the pillar pattern by explicitly handling the top faces. The lateral faces use the same encoding. However, the name *pyramid* describes the metaphorical process that is used to derive an implementation of the pattern: a virtual pyramid is embedded in the cuboid and slowly pushed toward and through the top. All cuboid fragments that intersect with the pyramid are assigned the latter, upcoming color. This intersection surface corresponds to the progress of the pillar pattern for lateral faces and results in a growing rectangle for top faces. As a result, this pattern encodes transition progress on both the lateral faces and the top face.

3.5 Noise

The noise pattern uses 3D noise (Perlin 2001) and a threshold to partition the surface for the two colors. This results in organic-looking surfaces and transition behavior. However, a suitable parameterization of the noise with respect to the scaling is challenging. We propose approximating the scale factor by the number of nodes in the tree as this measure loosely corresponds to the share of footprint area one node has in relation to the size of the whole treemap.

3.6 Dithering

The dithering pattern encodes the transition using a per-fragment dithering pattern. Dithering is applied in screen space regardless of the orientation of the cuboid. From a perceptual point of view, this results in a visual blending of the two colors (Ulichney 1988)—most likely, the medium on which this paper is read will

not be able to display the separated colors. However, we assert that only actual colors from a color scheme will be used using dithering.

3.7 Squares

The squares pattern extends the dithering by using a dithering pattern that can be scaled for the cuboid's size. The pattern is applied to the surface in world space, not the display pixels. Furthermore, it is directly affected by the cuboid's orientation. This reduces the visual blurring of colors while providing a transition nonetheless. From an implementation's point of view, designing the overhang from a square onto neighboring surfaces is challenging. A straightforward approach is to arrange the squares and loosen their aspect ratio to create rectangles such that no overhang occurs.

3.8 Arrows

This pattern generates the resemblance of arrows of alternating colors. The top faces behave similarly to the pillar pattern. In contrast to the pillar and pyramid patterns, the arrow direction can encode the direction of change in value, as the arrows can point upwards. Likewise, the pattern can be inverted to resemble downward facing arrows. These visual cues for the direction of change, however, prevents effective encoding of the direction of the animation. An extension to this pattern that includes the top face is presented with the arrows (full) pattern.

3.9 Arrows (full)

This pattern extends the arrow pattern by additional arrows on the top face. As with the arrow pattern, the arrows can be inverted, too, to encode the direction of change. However, the direction of the arrows on the lateral surfaces and the additional top surface allows for additional design considerations (Fig. 4). For one, they can be appended to the transition of the lateral faces, i.e., the transition starts with arrows on the lateral faces and ends with arrows on the top face. This allows encoding of direction on top only for a short duration of the transition. Alternatively, the arrows can be animated simultaneously on the top faces for the whole duration of the transition, whereby two variants with opposite directions are available. However, both directions, (1) the visual extension to the arrow-forming stripes from the lateral surfaces, and (2) individual arrows, pose non-intuitive reading along the lateral and top face border.

3.10 Pattern composition

Although some patterns can encode the semantics of direction of change, invariant to all patterns is that the direction of animation and the direction of change value can be encoded simultaneously. As an additional indicator for the direction of animation, we propose to superimpose two different patterns. One of the two distinctly colored areas is filled with a pattern instead of a solid color (Fig. 5). This allows for a direct, unambiguous display of change direction when paused and animated, as only one state has a second pattern superimposed. This pattern remains constant during the transition, e.g., reusing any other pattern with $\lambda = 1/2$. However, as the overall transition changes the surfaces encoding former and latter colors, the area of the secondary pattern will vary during the transition.

Candidates for Secondary Use. Technically, the secondary pattern is not limited by the choice of the main pattern. However, we suggest using a different, distinct pattern to the primary one, ideally one with a uniform distribution of colors. Combining both approaches ensures that the pattern is (1) distinguishable from the main pattern and (2) visible without regard to the current transition state. In Fig. 5, examples using the squares pattern, the noise pattern, and the arrow (full) pattern as secondary pattern are shown.

Parameterization of Secondary Pattern. The secondary pattern is derived for a static progress value $\lambda = 1/2$ and applied using a blending of the base color and a darker shade. This way, the pattern occupies half the area of the applied surface. Regarding the choice of the applied surface, we propose to use the surface that encodes the former color to superimpose the second pattern. The advantage of this is that the secondary pattern is only exposed (decreasingly) to the end of the transition, as the surface of the former color will vanish during the transition. Also, the secondary pattern should not be visible at the transition's beginning but should gradually fade in. Otherwise, the secondary pattern would appear abruptly at the beginning of a transition.

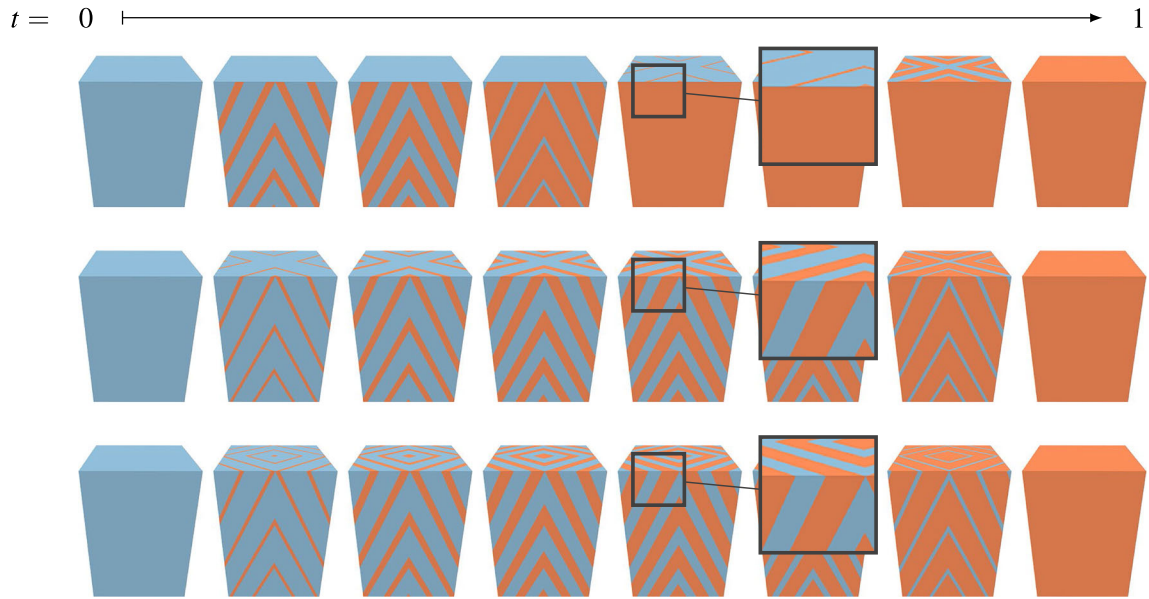


Fig. 4 Variants of the arrow-pattern transition for increasing values: lateral face first, then top face (top), or lateral and top faces simultaneously (center and bottom). Note that the bottom transition uses an inverted pattern for the top to indicate a value increase when seen from above

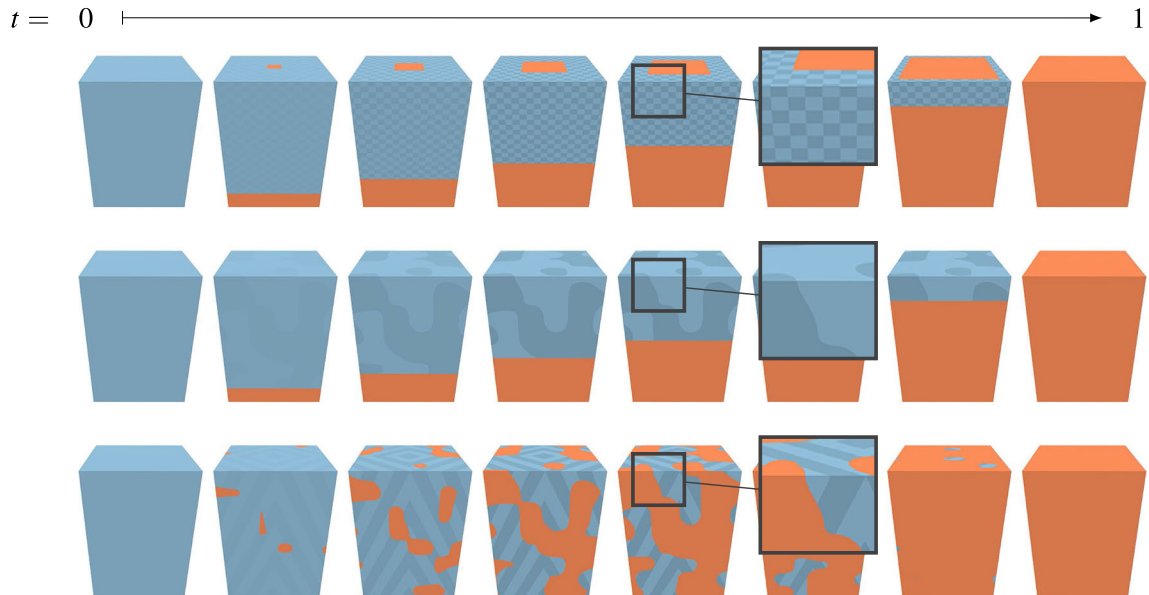


Fig. 5 Combination of two procedural patterns. The dominant pattern is supported by a secondary pattern placed only on the parts representing either the former (as can be seen here) or the upcoming value

3.11 Transition across multiple visual variables

The proposed procedural patterns allow two states of color on a cuboid's lateral and top surfaces. However, additional visual variables are usually used with a treemap, especially a 2.5D treemap. For example, the node weight is mapped to the footprint area, and another attribute is mapped to height (Fig. 6). With these

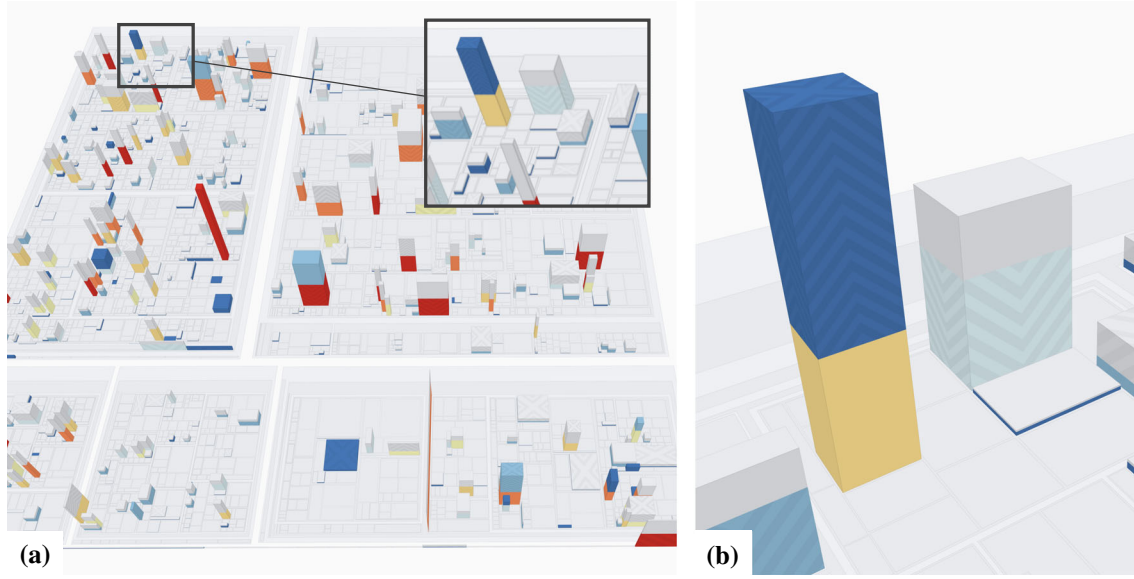


Fig. 6 A software map within the middle of an animated transition of two revisions of data **a** mapped to height and color. For change display, the Pillar pattern is used with the Arrow (full) pattern as the secondary pattern. The detail view **b** shows two different changes. First, one can see that the left cuboid is growing, and its color is changing from blue to yellow. Second, the right cuboid is shrinking to zero, while the color it changes to is only used for deleted or irrelevant nodes

additional mappings, a transition should include the changes among all mapped visual variables (Fig. 7). Especially for the height mapping, we see visual interference with most of the proposed patterns that come into play when an animated transition is presented to the user. Further, some patterns visually encode a direction that can be used to encode the direction of change of other visual variables, such as height. An example pattern for this is the arrow pattern.

3.12 Transitions and animations

So far, we have considered isolated transitions from one color to another for a single node. However, we further consider using animations and animation control to display multiple changes over a whole treemap and multiple snapshots of data.

Transition. A transition describes a change from a node's state or mapped value to another. Thereby, a transition will not encode time or duration but only progress. The required parameter for such a transition is its progress control $t \in [0, 1]$ where 0 is used at the start and 1 at the end of a transition. This control value t is then mapped to the pattern progress value λ . In our case, we use the identity mapping $\lambda = t$, but this can be fine-tuned using other mappings as well. Next to easing functions for aesthetic and user-experience purposes (Thomas and Johnston 1981), this mapping can be used to account for human perception of area ratios.

Animation. We use animation as the continuous progression over multiple states of a dataset using transitions. More important, it is linked to the concept of time and duration. The animation between two points in time, e.g., two subsequent revisions for software system data, can range from running all transitions for every node and every visual variable in parallel to a strict sequence. The number of changes between two dataset states severely limits the first approach's usability. If there are many changes or if the changes are highly scattered, a user will only be able to grasp individual changes during an animation. The number of simultaneous transitions can be controlled with explicit management of individual transitions. For example, similar changes can be grouped, e.g., due to the same semantics or proximity within the treemap, and different changes can be ordered one after the other. The explicit management is implemented by different starting points and transition durations for each node. We apply different constraints to the configurations of an animation:

- separate animations from different visual variables, e.g., area, height, and color,
- order decreases in values before increases in values, and

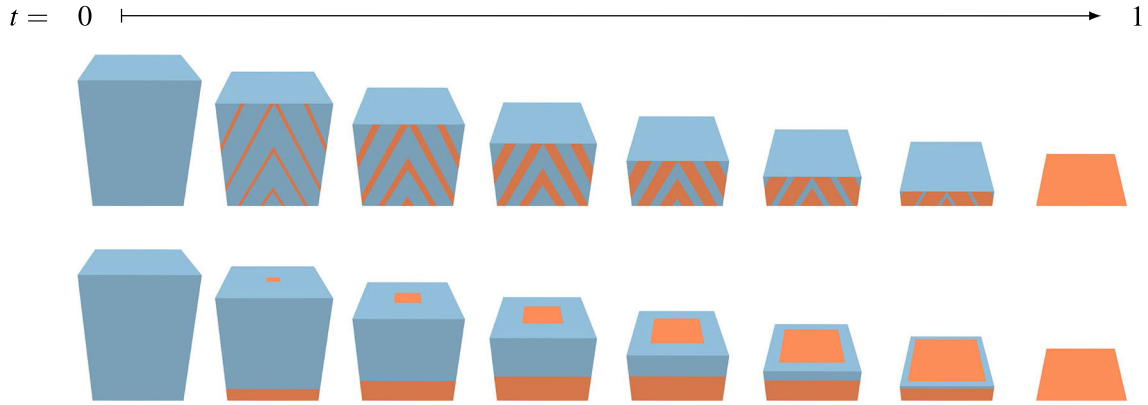


Fig. 7 An example of two patterns—Arrows (top) and Pyramid (bottom)—that are used to depict the simultaneous increase in color and decrease in height

- separate animations based on proximity in the treemap.

A configured animation from one state of a dataset to another can be handled as follows: a virtual, global time advances, and for each time-step, the per-node transition control values are updated, and the treemap is redrawn. This process ends as soon all transition are completed.

4 Evaluation

We evaluate the transition-aware patterns by applying them to software system data and a comparison of characteristics of the patterns. The prototypical implementation is written in TypeScript and GLSL. The rendering component is based on WebGL and the rendering framework `webgl-operate`. We assessed the procedural texture patterns, using them individually as well as in combination, and applied them for color change emphasis, including height and weight changes on small and large real-world datasets. For these we used (1) an open-source project and (2) an internal, larger software project. The results indicate the technical feasibility of the patterns, animation system, and rendering approach. However, they serve only as a starting point for much-needed user studies regarding readability of change (Fiedler et al. 2020).

4.1 Implementation

Our prototype is an extension of a WebGL-based visualization system for large-scale 2D and 2.5D treemaps by a dual-state implementation. We use the concept of a union tree to represent two states—a former and a latter state—of a tree-structured dataset within the visualization system (Tu and Shen 2007). The system supports area, height, and color mapping as visual variables. Specific for our prototype, we added an animation controller, adapted the geometry representation, and added pattern computation to the rendering of the cuboids.

4.2 Animation controller

For the animation in general, we had to redesign large parts of our visualization and rendering implementation. We composed the handling of animations and transitions to a system that manages all temporal animations, controlling the time-dependent transition values and the individual per-node transition states (Fig. 8). Our prototype allows for independent, simultaneous transitions per node for every mapped data attribute. The procedural patterns are implemented as temporal visual variables, mapping a 3-tuple (a value per state and transition progress) to color. An animation of weight and height mappings is implemented similarly. The latter allows for ordering and masking of transitions based on (1) transition type, (2) user-defined and node-associated values, (3) metadata, (4) enumeration, and more.

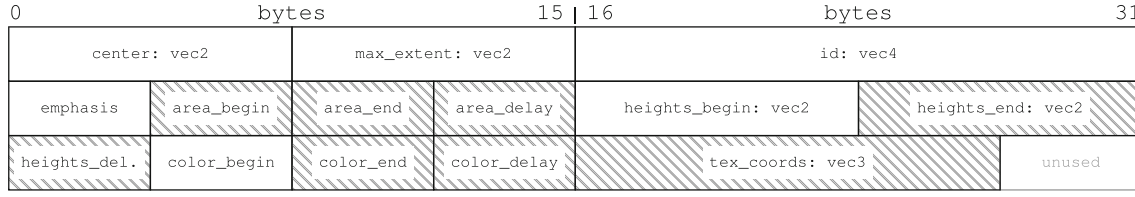


Fig. 8 Illustration of the memory layout of the per-cuboid data. The cell’s width corresponds to the memory consumption; the base width corresponds to one float, i.e., 4 bytes, and multiples of two, three, and four corresponds to the vector formats vec2, vec3, and vec4, respectively. The memory representation thus consumes 92 bytes per cuboid. This per-cuboid data allows representing both a former state and a latter state for area mapping, height mapping, and color mapping. Further, each mapping has an individual delay value for each cuboid to allow fine-grained animation control. The *emphasis* affects the kind of visual emphasis on the node, e.g., outlines or highlighting fill color. The pattern indicates the per-cuboid data that was added to support animated transitions. The per-cuboid data for a static display consumes 48 bytes

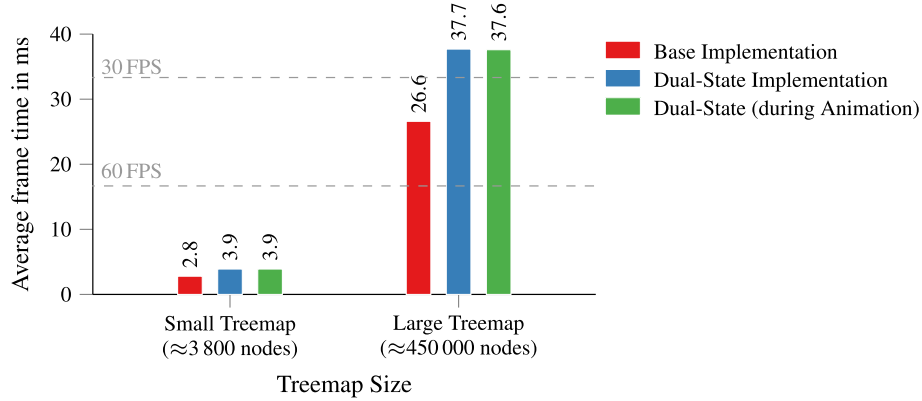


Fig. 9 Performance analysis on a small treemap and a large treemap example based on the base implementation and the dual-state implementation, as well as an additional measure for the dual-state implementation during an active animation. The system for the measurements was a Ryzen 5 1600X CPU with 16GB RAM and an Nvidia GTX 1060 GPU with 6GB VRAM. The target resolution for rendering was FullHD (1080×1920 pixels). Each measure is the average of 32 000 frame renderings

4.3 Geometry memory layout

The geometry is based on the concept of attributed vertex clouds: Although the geometry of a cuboid has multiple vertices and corresponding polygon mesh as volatile geometry, the data for attribute mapping are represented once in GPU memory (Scheibel et al. 2017). We extended this memory layout to include area, height, and color values for both the former and the latter state (Fig. 8). The rendering pipeline can compute a node’s corresponding position, extent, and procedural texture pattern based on the per-node progress value.

4.4 Pattern computation

The implementation uses fragment shaders and consists of the actual pattern computation and a blending with the other aspects of the shading (Listing 1, Listing 2). Implementing the patterns themselves was straightforward and imposed no significant impact on rendering performance (runs on mobile devices for large treemaps up to 100 000 nodes). For desktop machines, the performance impact was around 30% compared to the base implementation (Fig. 9). The patterns are implemented using *signed distance functions* (Vivo and Lowe 2015) and fragment shaders. The patterns differ mainly in appearance, granularity, and intended surfaces, but they share a constant run-time overhead.

```

1  struct PatternConfig {
2      int mode; // type of pattern, e.g., pillar
3      float progress; // node-local progress value
4  };

6  struct PatternData {
7      int face; // the current axis-oriented face of a cuboid
8      float nodeCountSqrt; // pattern scale factor
9      vec2 texCoord2D; // face-local 2D coordinates
10     vec3 texCoord3D; // cuboid-local 3D coordinates
11     vec3 uv; // pattern-adjusted local 3D coordinates
12     vec4 worldPos; // treemap-global 3D coordinates
13     float relativeHeight; // relative height of cuboid
14     float absoluteHeight; // absolute height of cuboid
15 };

17 PatternData patternData = PatternData(
18     u_face, u_nodeCountSqrt, v_texCoord2D, v_texCoord3D,
19     v_uv, v_worldPos, v_relativeHeight, v_absoluteHeight
20 );

22 PatternConfig diffPatternConfig = PatternConfig(
23     u_diffPatternMode, 0.5
24 );

26 PatternConfig colorConfig = PatternConfig(
27     u_colorMode, v_colorProgress
28 );

30 // returns distance to border between base color and target color.
31 // < 0 represents base color
32 // >= 0 represents target color
33 float calculateColorDistance(in PatternData data, in PatternConfig config);

35 void main() {
36     // ...
37     float baseColorDistance =
38         calculateColorDistance(patternData, diffPatternConfig);
39     float baseColorDecider = smoothstep(0.0, 0.5, v_colorProgress)
40         * step(0.0, baseColorDistance);
41     float colorDistance = calculateColorDistance(patternData, colorConfig);
42     float colorDecider = step(0.0, colorDistance);
43     vec4 colorBegin =
44         mix(v_baseColorBegin, v_patternColorBegin, baseColorDecider);
45     fragColor = mix(baseColorBegin, v_baseColorEnd, colorDecider);
46 }

```

Listing 1 Excerpt of the GLSL fragment shader for shading treemap cuboids with patterns of the proposed pattern catalog. Secondary patterns are considered as well. The actual implementation of a pattern resides in `calculateColorDistance`, which is shown in Listing 2.

```

1 // returns distance to border between base color and target color.
2 // < 0 represents base color
3 // >= 0 represents target color
4 float calculateColorDistance(in PatternData data, in PatternConfig config) {
5     // vertical pattern
6     if (config.mode == 0) {
7         return config.progress - data.relativeHeight;
8     }
9     // ...
10
11     // default
12     return 0.0; // means base color
13 }

```

Listing 2 Example implementation of the *vertical* pattern in GLSL. Listing 1 contains the definitions for *PatternData* and *PatternConfig*.

4.5 Case study

Based on the software prototype, we applied the texture patterns on a software system dataset created from the structure and measured software metrics on the Open Source C++ project *cpplocate*¹. The source code was measured for the metrics *lines of code*, *nesting level*, *cyclomatic complexity*, *number of developers*, and *change frequency*. For this example, the former state was set to October 27, 2018, while the latter state for analysis was March 2, 2019. The used mappings are *lines of code* for area mapping, *cyclomatic complexity* for height mapping, resulting in an abstract volume of implementation for the volume of the cuboid of the treemap, and the *change frequency* for color mapping.

The visualization highlights that only parts of the software system changed during this period, which is expected for software development (Fig. 10). Further, it shows a decrease in change frequency (mapped to color), and cyclomatic complexity (mapped to height) for the source code module in focus. The change in the color attribute could be encoded in the procedural texture pattern, while the change in the height attribute was only incidentally decreasing, too.

In comparison, we also applied the visualization to a closed-source software system dataset with above 450 000 nodes (Fig. 11). The number of nodes results in a small screen space for average-sized nodes. As a result, changes on small and average-sized nodes get overlooked. Regarding run-time performance, adding the procedural texture patterns had a noticeable effect (Fig. 9). The average frame time dropped below 30 frames per second but still allowed for animated transitions.

4.6 Discussion

We applied the texture patterns for mid-scale and large software projects, which is only a small share of application scenarios for treemaps and 2.5D treemaps. However, we see the software analytics domain as suitable for our analysis because (1) the number of changes between two revisions is usually small, and (2) it allows to test the scalability of visualization technique approaches towards hundreds of thousands of items. As a result, we see the texture patterns are visible for small and mid-sized software projects but get harder to detect for large software projects, even with animated transitions. Although procedural texture patterns to encode a transition of color is a straightforward approach, integrating with other visual features of a 2.5D treemap remains challenging.

4.7 Pattern characteristics

Defining a pattern that fulfills all proposed characteristics remains an open task. We proposed seven patterns with different characteristics, but none of them fulfills all of the encoding and capability characteristics (Table 1). However, we want to highlight the arrows (full) pattern that supports most encoding characteristics, as seen in the $\sum_{\nabla E}$ score. Regarding both encoding and compatibility characteristics

¹Par55 Source code hosted on GitHub at <https://github.com/cginternals/cpplocate>, which, at the time of measurement, still included *googletest* as an in-source module.

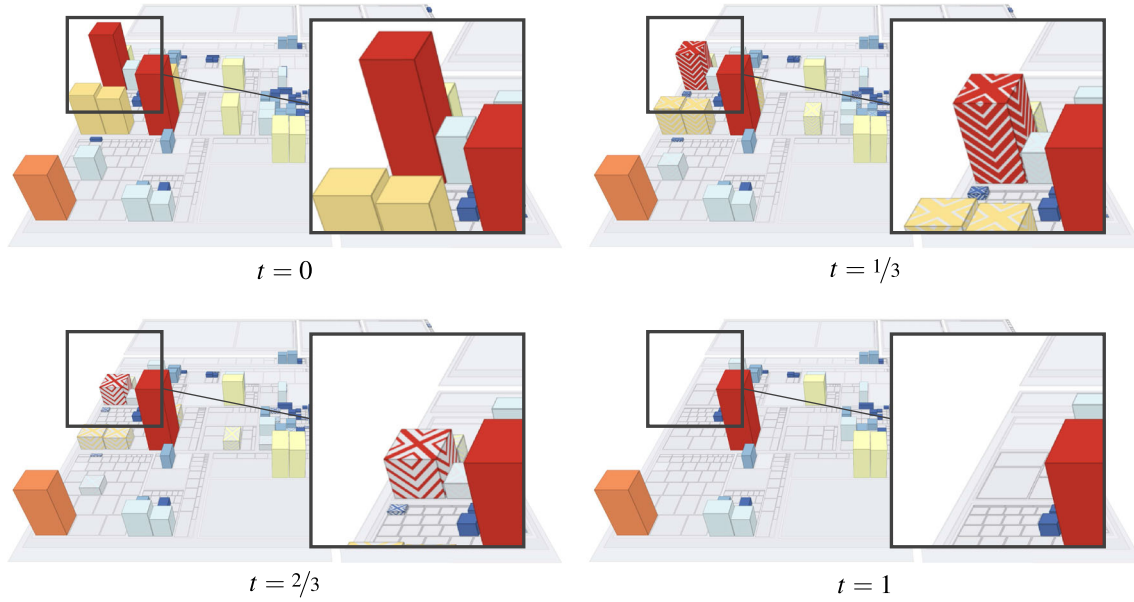


Fig. 10 Four snapshots with progress values of 0, $1/3$, $2/3$, and 1 of an animated transition between the state of the software project *cpplocate* from October 27, 2018, to March 2, 2019

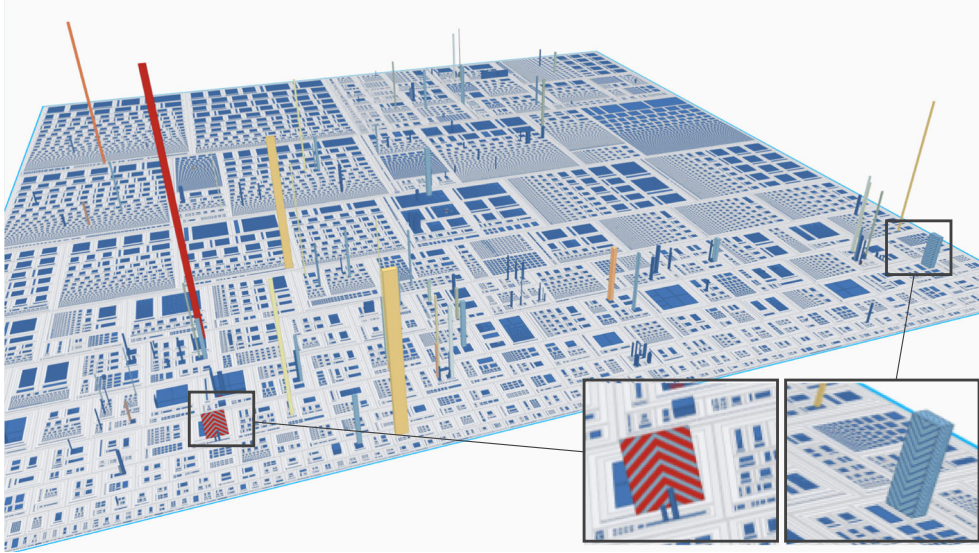









Fig. 11 Software map of an internal software project with above 450 000 nodes. It is displayed within the middle of an animated transition of two revisions of data. For change display, the *Arrow* pattern is used for increasing values and an inversed *Arrow* pattern is used for decreasing values

unweighted, using the sum of $\sum_{\nabla E}$ and $\sum_{\nabla C}$, the Dithering pattern is the most versatile. By design, all of our proposed pattern are monotonic and thus support the $\nabla_{Monotonic}$ characteristic. Regarding the $\nabla_{DirStates}$ characteristic, only the pillar and pyramid pattern support the direct encoding of the former and latter state through orienting the old value above the new value. Here, an explicit encoding using a secondary pattern allows encoding former and latter states for all other patterns. For the $\nabla_{DirChange}$ characteristic, only the arrow and the arrow (full) pattern allow encoding the direction of change. However, their arrow pattern can encode other increases and decreases, such as change in height, as well. For example, the arrow was previously used to encode change in height (Limberger et al. 2019a), as shown in Fig. 2. This further highlights the challenges to encode multiple changes in values on one treemap node. Almost all patterns support the $\nabla_{RatioLat}$ characteristic. However, the noise pattern has the disadvantage that designing a noise

Table 1 Overview of the patterns and their encoding and compatability characteristics. This overview is based on technical limitations on a feature level and poses hard constraints for the use of proposed patterns. There might be additional limitations imposed by user expectations or user experience. The scores $\sum_{\nabla C}$ and $\sum_{\nabla E}$ are the weighted sum of the respective characteristics. A supported characteristic count as 1, a partially supported characteristic as 0.5 and an unsupported characteristic as 0. Legend: ● supported | ○ partial support | – unsupported

Procedural Pattern	Encoding – ∇E						Compatibility – ∇C				
	$\nabla_{\text{Monotonic}}$	$\nabla_{\text{DirStates}}$	$\nabla_{\text{DirChange}}$	∇_{RatioLat}	∇_{RatioTop}	$\sum_{\nabla E}$	$\nabla_{\text{IndHeight}}$	$\nabla_{\text{IndWeight}}$	∇_{TreeSize}	$\nabla_{\text{Polygonal}}$	$\sum_{\nabla C}$
 Pillar	●	●	–	●	–	3	○	●	●	●	3.5
 Pyramid	●	●	–	●	○	3.5	○	●	●	○	3
 Dithering	●	–	–	●	●	3	●	●	●	●	4
 Arrows	●	–	●	●	–	3	●	●	○	○	3
 Arrows (full)	●	–	●	●	●	4	●	●	○	–	2.5
 Noise	●	–	–	○	○	2	●	●	○	●	3.5
 Squares	●	–	–	●	●	3	●	●	○	○	3

function that matches the ratio of both colors to the progress value is challenging. The ∇_{RatioTop} characteristic is supported by fewer patterns. With two of them, the pillar and arrow patterns, this is not supported as the top face is not part of the transition. In addition, the pyramid pattern and the noise pattern only partially provide this characteristic as the ratio of colors is skewed to the current progress value. Almost all proposed patterns support the characteristics regarding independence from the height and weight value changes ($\nabla_{\text{IndHeight}}$ and $\nabla_{\text{IndWeight}}$). However, regarding height, the pillar and pyramid patterns are an exception. For both, the border between the colors depends on the cuboid’s height. It thus moves nonlinearly when both color and height are animated simultaneously. We see the applicability for large treemaps (∇_{TreeSize}) given with the pillar, pyramid, and dithering patterns. The other patterns are applicable in a general sense but suffer in situations of little screen space. The applicability of the patterns for non-rectangular treemaps and general 2.5D visualizations is captured in the $\nabla_{\text{Polygonal}}$ characteristic. The support is mixed across the proposed patterns. However, the proposed patterns were designed with rectangular treemaps in mind, but other patterns for non-rectangular extruded shapes can be designed, too.

4.8 Combination of patterns

Combining the proposed texture patterns with other visual features is expected and supported. However, we expect any combined use to influence the used visual mappings. For example, visualizing change in color and another visual variable such as height limit the choice of visual mappings considerably (Fig. 7). Combining multiple procedural patterns through the secondary pattern may allow distinguishing between former and latter states. However, another color is introduced that conflicts with the applied color scale.

4.9 Implementation

Implementing the prototype using hardware-accelerated rendering and procedural texture patterns seems advantageous, as it allows for a broad range of patterns, combinations, and extensions. Using a pattern as a starting point is straightforward (see Fig. 12). The overhead in rendering times is reasonable and allows for interactive frame rates for mid-scale and even large datasets with hundreds and thousands of nodes.

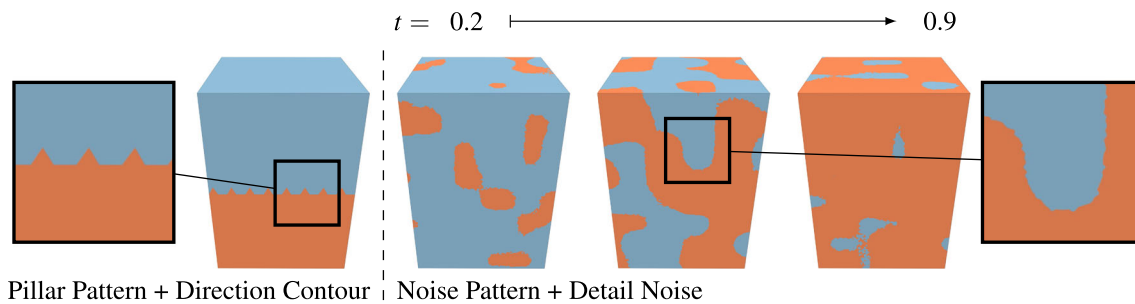


Fig. 12 Example of two extensions to the proposed pillar pattern and Noise pattern. The pillar pattern was extended by a direction contour (left) that uses indents in the border between the two colors resembling arrowheads to encode the animation direction. The noise pattern was extended with detail noise (right). However, this detail noise does not allow encoding a direction of change

5 Conclusions

We have shown that exploring time-varying, tree-structured data using 3D-embedded treemaps is feasible by means of (and in addition to area, height, and color) transitional visual variables. When depicting two consecutive states in an in situ fashion, encoding the direction of change is crucial to decode the visualization with its former and latter state effectively. We proposed texture patterns that support displaying changes in colors using procedural textures. Those texture patterns are applicable for static contexts—i.e., static images—, or interactive contexts through interactive clients with navigation and interaction. For specific templates on how to transition color values in 3D we proposed the seven variants (1) Pillar, (2) Pyramid, (3) Arrows, (4) Arrows (full), (5) Noise, (6) Dithering, and (7) Squares. We discussed the different characteristics, especially the direction of change in value and the direction of animation, and the strengths and weaknesses of each pattern. Due to a pending user evaluation, we cannot conclude a default pattern at this point. Further, the proposed texture patterns are currently not fully compatible with other approaches using procedural textures used to encode the direction of change for spatial changes of cuboids. The prototype indicates that these procedural texture patterns require little overhead for basic implementations, and interactive framerates are achieved even for large-sized treemaps.

Future work includes quantitative evaluations using comprehensive user studies and extended case studies on large software systems and long periods. The goal would be to find one pattern or a small set of patterns that perform best regarding readability. Further, the proposed patterns can be revised and extended, e.g., to support general extruded shapes for 2.5D treemaps and 2.5D information visualization in general. Another goal is to find combinations of different, independent encodings for the direction of change for areas, heights, and colors.

Acknowledgements We want to thank the anonymous reviewers for their valuable comments and suggestions to improve this article. This work is part of the “Software-DNA” project, which is funded by the European Regional Development Fund (ERDF or EFRE in German) and the State of Brandenburg (ILB). This work is also part of the ZIM project “TASAM”, which is funded by the German Federal Ministry for Economic Affairs and Energy (BMWi).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

- Benomar O, Sahraoui H, Poulin P (2013) Visualizing software dynamicities with heat maps. In: 2013 1st IEEE working conference on software visualization, VISSOFT ’13, pp 1–10, <https://doi.org/10.1109/VISSOFT.2013.6650524>

- Bladh T, Carr DA, Kljun M (2005) The effect of animated transitions on user navigation in 3d tree-maps. In: Proceedings of the 9th international conference on information visualisation, IV '05, pp 297–305, <https://doi.org/10.1109/TV.2005.122>
- Carpendale MST (2003) Considering visual variables as a basis for information visualisation. Tech. Rep. 2001-693-16, University of Calgary, <https://doi.org/10.11575/PRISM/30495>
- Caserta P, Zendra O (2011) Visualization of the static aspects of software: a survey. *IEEE Trans Visual Comp Graph* 17(7):913–933. <https://doi.org/10.1109/TVCG.2010.110>
- Chen Y, Du X, Yuan X (2017) Ordered small multiple treemaps for visualizing time-varying hierarchical pesticide residue data. *Vis Comp* 33(6):1073–1084. <https://doi.org/10.1007/s00371-017-1373-x>
- Fiedler C, Scheibel W, Limberger D, Trapp M, Döllner J (2020) Survey on user studies on the effectiveness of treemaps. In: Proceedings of the 13th International symposium on visual information communication and interaction, ACM, VINCI '20, pp 2:1–10. <https://doi.org/10.1145/3430036.3430054>
- Hagh-Shenas H, Kim S, Interrante V, Healey C (2007) Weaving versus blending: a quantitative assessment of the information carrying capacities of two alternative methods for conveying multivariate data with color. *IEEE Trans Visualiz Comp Graph* 13(6):1270–1277. <https://doi.org/10.1109/TVCG.2007.70623>
- Johnson BS, Shneiderman B (1991) Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In: Proceedings of the 2nd conference on visualization '91, IEEE, VIS '91, pp 284–291, <https://doi.org/10.1109/VISUAL.1991.175815>
- Kindlmann G, Scheidegger C (2014) An algebraic process for visualization design. *IEEE Trans Visual Comp Graph* 20(12):2181–2190. <https://doi.org/10.1109/TVCG.2014.2346325>
- Langelier G, Sahraoui H, Poulin P (2008) Exploring the evolution of software quality with animated visualization. In: Proceedings of the symposium on visual languages and human-centric computing, IEEE, VLHCC '08, pp 13–20, <https://doi.org/10.1109/VLHCC.2008.4639052>
- Limberger D, Matthias T, Döllner J (2019a) In-situ comparison for 2.5d treemaps. In: Proceedings of the 14th International joint conference on computer vision, imaging and computer graphics theory and applications – Volume 3: IVAPP, INSTICC, SciTePress, IVAPP '19, pp 314–321, <https://doi.org/10.5220/0007576203140321>
- Limberger D, Scheibel W, Trapp M, Döllner J (2019b) Advanced visual metaphors and techniques for software maps. In: Proceedings of the 12th International symposium on visual information communication and interaction, ACM, VINCI '19, pp 11:1–8, <https://doi.org/10.1145/3231622.3231638>
- Limberger D, Scheibel W, van Dieken J, Döllner J (2021) Visualization of data changes in 2.5d treemaps using procedural textures and animated transitions. In: Proceedings of the 14th International symposium on visual information communication and interaction, ACM, VINCI '21, pp 6:1–5, <https://doi.org/10.1145/3481549.3481570>
- L'Yi S, Jo J, Seo J (2021) Comparative layouts revisited: design space, guidelines, and future directions. *IEEE Trans Visualiz Comp Graph* 27(2):1525–1535. <https://doi.org/10.1109/TVCG.2020.3030419>
- Perlin K (2001) Noise hardware. In: real-time shading SIGGRAPH course notes (2001), ACM, chap 2
- Quilez I (2013) Filtering procedural textures. <https://www.iquilezles.org/www/articles/filtering/filtering.htm>
- Roberts RC, Laramée RS (2018) Visualising business data: a survey. *MDPI Information* 9(11), <https://doi.org/10.3390/info9110285>
- Scheibel W, Trapp M, Döllner J (2016) Interactive revision exploration using small multiples of software maps. In: Proceedings of the 11th Joint conference on computer vision, imaging and computer graphics theory and applications – Volume 2: IVAPP, INSTICC, SciTePress, IVAPP '16, pp 131–138, <https://doi.org/10.5220/0005694401310138>
- Scheibel W, Buschmann S, Trapp M, Döllner J (2017) Attributed Vertex Clouds, Bowker Identifier Services, pp 3–21
- Scheibel W, Limberger D, Döllner J (2020) Survey of treemap layout algorithms. In: Proceedings of the 13th international symposium on visual information communication and interaction, ACM, VINCI '20, pp 1:1–9, <https://doi.org/10.1145/3430036.3430041>
- Schulz HJ, Hadlak S, Schumann H (2011) The design space of implicit hierarchy visualization: a survey. *IEEE Trans Visualiz Comp Graph* 17(4):393–411. <https://doi.org/10.1109/TVCG.2010.79>
- Thakur S, Rhyne TM (2009) Data vases: 2d and 3d plots for visualizing multiple time series. In: Advances in visual computing: 5th international symposium, Part II, Springer, ISVC '09, pp 929–938, https://doi.org/10.1007/978-3-642-10520-3_89
- Thomas F, Johnston O (1981) Disney animation: the illusion of life. Abbeville Press
- Tu Y, Shen HW (2007) Visualizing changes of hierarchical data using treemaps. *IEEE Trans Visualiz Comput Graph* 13(6):1286–1293. <https://doi.org/10.1109/TVCG.2007.70529>
- Ulichney RA (1988) Dithering with blue noise. *Proceed IEEE* 76(1):56–79. <https://doi.org/10.1109/5.3288>
- Urness T, Interrante V, Marusic I, Longmire E, Ganapathisubramani B (2003) Effectively visualizing multi-valued flow data using color and texture. In: Proceedings of the 2003 International conference on visualization, IEEE, VIS '03, pp 115–121, <https://doi.org/10.1109/VISUAL.2003.1250362>
- Vernier E, Sondag M, Comba J, Speckmann B, Telea A, Verbeek K (2020) Quantitative comparison of time-dependent treemaps. *EG Comp Graph For* 39(3):393–404. <https://doi.org/10.1111/cgf.13989>
- Vivo PG, Lowe J (2015) The book of shaders. <https://thebookofshaders.com/>
- Wettel R, Lanza M (2008) Visual exploration of large-scale system evolution. In: Proceedings of the 15th working conference on reverse engineering, IEEE, WCRE '08, pp 219–228, <https://doi.org/10.1109/WCRE.2008.55>
- Würfel H, Trapp M, Limberger D, Döllner J (2015) Natural Phenomena as Metaphors for Visualization of Trend Data in Interactive Software Maps. In: Proceedings of the conference on computer graphics & visual computing, CGVC '15, pp 69–76, <https://doi.org/10.2312/cgvc.20151246>